1, session(理解)

- 目的: 如何配置session的存储位置, 通过request对象获取,设置
- 操作:
  - 1, 指定redis存储,来自于django-redis文档;

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    }
}
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
SESSION_CACHE_ALIAS = "default"
```

  - 2, 操作方法
    - 设置: request.session(key) = value
    - 获取: request.session.get(key)
    - 移出:request.session.pop(key,None)
    - 清空:request.session.flush()
    - 有效期: request.session.set_expiry(time)
- 注意点:
  - 1,如果不做任何配置session默认存储在sqlite数据库
    - 需要进行迁移才可以看到效果

2,类视图(掌握)

- 目的: 能够定义类视图,并且指定路由,访问
- 使用步骤:
  - 1, 定义类,继承自View,里面的方法需要以标准的请求方式来命名

```python
from django.views import View
#2,类视图
class PersonView(View):

    def get(self,request):
        return HttpResponse("get请求")

    def post(self,request):
        return HttpResponse("post请求")
```

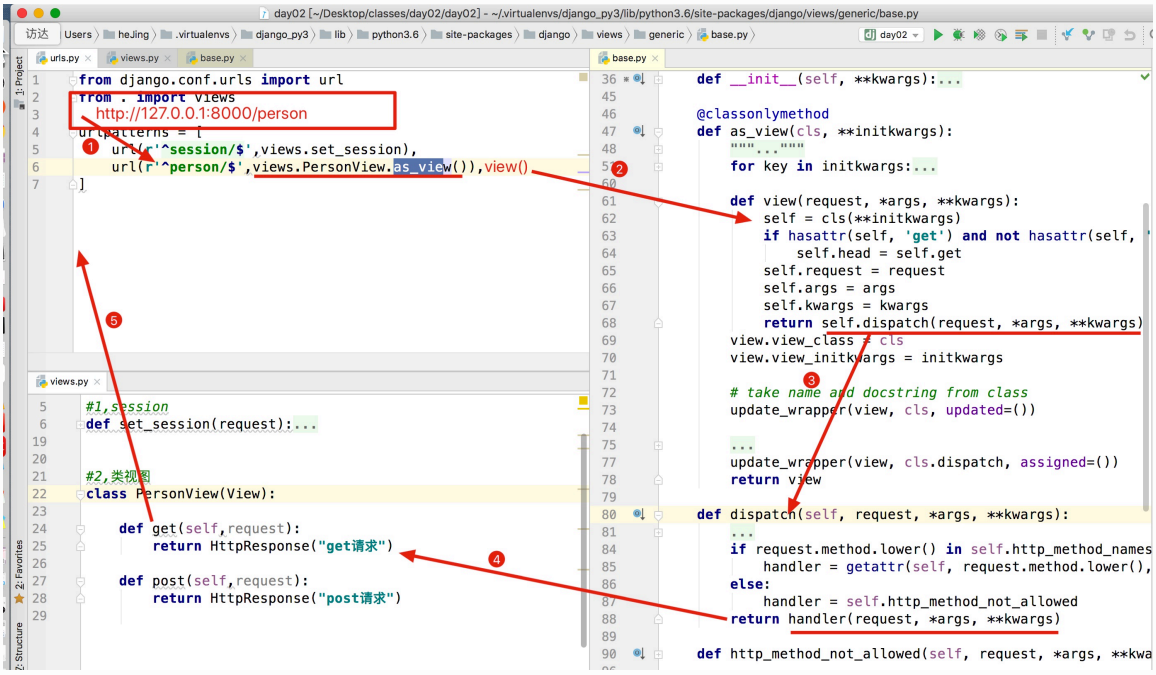
- 2,编写路由

```python
url(r'^person/$',views.PersonView.as_view()),
```

- 好处:
  - 1, 可读性更好
  - 2, 可以使用面向对象的思想, 通过继承提高代码的复用性

3,类视图原理(理解)

- 目的: 了解调用过程
- 调用流程图解:



3,类视图增加装饰器(了解)

- 目的: 能够使用提供的两种方式, 给类增加额外功能
- 方式一: 在路由上面装饰

```
#booktest/views.py
def user_login_data(view_func):
    def wrapper(request,*args,**kwargs):

        print("给%s增加了额外的功能"%request.method)

        return view_func(request,*args,**kwargs)

    return wrapper

# booktest/urls.py
url(r'^register/$',user_login_data(views.RegisterView.as_view()
))
```

- 方式二: 使用系统提供的method_decorator; (推荐使用,更加灵活)

```
from django.utils.decorators import method_decorator

# @method_decorator(user_login_data,name="dispatch") #
给RegisterView2里面所有的方法都添加
# @method_decorator(user_login_data,name="get")
class RegisterView2(View):

    @method_decorator(user_login_data)
    def get(self,request):
        return HttpResponse("RegisterView2 get请求")

    def post(self,request):
        return HttpResponse("RegisterView2 post请求")
```

4,中间件(了解)

- 目的: 知道中间件中的代码和视图函数中的代码, 执行的时间顺序

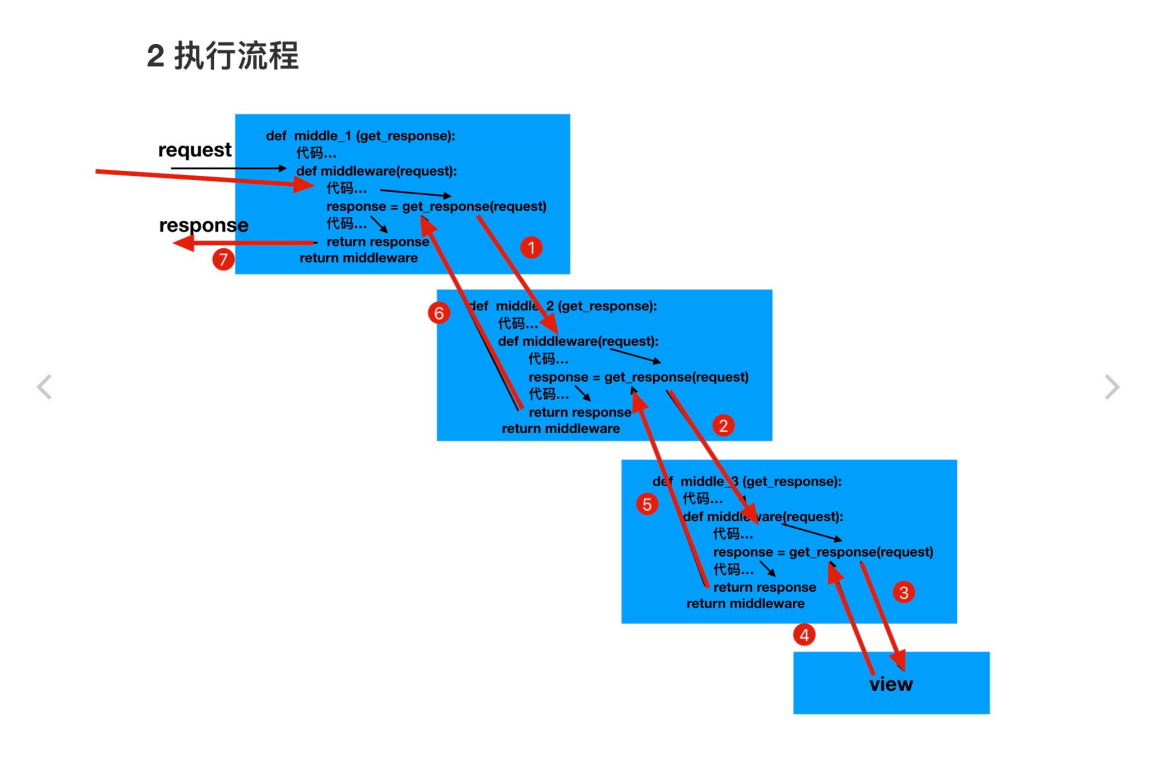
- 使用流程:
  - 1, 创建文件,定义中间件方法; booktest/my_middleware.py

```
def simple_middleware(get_response):
    # 此处编写的代码仅在Django第一次配置和初始化的时候执行一次。
    print("1, init")
    def middleware(request):
        # 此处编写的代码会在每个请求处理视图前被调用。
        print("1, before_request")

        response = get_response(request)

        # 此处编写的代码会在每个请求处理视图之后被调用。
        print("1, after_request")

        return response

    return middleware
```

  - 2,注册中间件列表; settings.py

```
MIDDLEWARE = (
    ...
    'booktest.my_middleware.simple_middleware',
)
```

- 调用流程图解



5,模板(了解)

- 目的: 模板的渲染, 以及基本模板语法的使用
- 使用格式:
- 1, 设置模板文件夹, settings.py

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,"templates")],
        ...
    },
]
```

- 2, 渲染页面

```
class TemplateView(View):
    def get(self,request):
        #方式一: 完整方式
        # template = loader.get_template('file01.html')
```

```
    #
    # file_data = template.render(context={"name":"zhagnsan"})
    #
    # response = HttpResponse(file_data)

    # return response

    #方式二: 简化格式
    context = {
        'city': '北京',
        'adict': {
            'name': '西游记',
            'author': '吴承恩'
        },
        'alist': (1, 2, 3, 4, 5)
    }
    return render(request,'file01.html',context=context)
```

- 3,使用模板语法
    - 获取变量, for循环, if分支, 过滤器, 模板继承(参考文档)

6,数据库配置(掌握)

- 目的 : 能够配置数据库的链接信息
- 配置流程:
    - 1, 配置数据库的连接信息(看文档) settings.py

```
DATABASES = {
    'default': {
        # 'ENGINE': 'django.db.backends.sqlite3',
        # 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), #不用,小型数据库

        #mysql的配置
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django1',
        'USER': 'root',
```

```
        'PASSWORD': '123456',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

- 2,设置数据库驱动; day02/init.py

```
import pymysql
pymysql.install_as_MySQLdb()
```

- 3,创建数据库

7,模型类图书(理解)

- 目的: 理解模型类中的字段含义,能够定义出对应的模型类
- 格式:

```
class Person(models.Model):
    name = models.字段类型(约束)
    字段类型: 参考文档
    约束: 参考文档
```

- 具体模型类

```
#1,编写图书模型类(一方)(id,书名,发布日期,阅读量,评论量,是否删除)
class BookInfo(models.Model):
    btitle = models.CharField(max_length=20,verbose_name="书名")
    bpub_date = models.DateField(verbose_name="发布日期")
    bread = models.IntegerField(verbose_name="阅读量",default=0)
    bcomment = models.IntegerField(verbose_name="评论量",default=0)
```

```
    is_delete = models.BooleanField(verbose_name="逻辑删除",default=False)

    #指定表的信息
    class Meta:
        db_table = "tb_books" #指定表名

    #输出对象的时候,显示内容
    def __str__(self):
        return self.btitle
```

8,模型类英雄(理解)

- 目的: 理解模型类中的字段含义,能够定义出对应的模型类
- 模型类

```
#2,英雄模型类(多方)
class HeroInfo(models.Model):
    GENDER_CHOICES = (
        (0, 'female'),
        (1, 'male')
    )
    hname = models.CharField(max_length=20, verbose_name='名称')
    hgender = models.SmallIntegerField(choices=GENDER_CHOICES, default=0, verbose_name='性别')
    hcomment = models.CharField(max_length=200, null=True, verbose_name='描述信息')
    #on_delete=models.CASCADE, 删除书籍的时候,将其下面的所有英雄也删除
    hbook = models.ForeignKey(BookInfo, on_delete=models.CASCADE, verbose_name='图书') # 外键
    is_delete = models.BooleanField(default=False, verbose_name='逻辑删除')

    class Meta:
        db_table = 'tb_heros'

```

```
    #输出对象的时候,显示内容
    def __str__(self):
        return self.hname
```

9,数据库迁移,添加测试数据(掌握)

- 目的: 能够将编写模型类迁移到数据库中
- 操作流程:
    - 1, 将子应用注册到INSTALLED_APP

```
INSTALLED_APPS = (
    ...
    'booktest.apps.BooktestConfig'
)
```

    - 2,迁移
        - 生成迁移脚本: python manage.py makemigrations
        - 执行迁移脚本: python manage.py migrate
    - 3,添加测试数据
        - source 提供的sql文件

10,数据库日志信息查看(理解)

- 目的: 了解ORM本质是生成的sql语句
- 查看流程:
    - 1, 配置mysql的配置文件, 放开日志, 监听日志

```
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
放开: 68,69行

重启: sudo serivce mysql restart

监听: sudo tail -f /var/log/mysql/mysql.log

```

    - 2, 准备好对应的数据库
    - 3, 进入到终端中,通过ORM类操作数据库
        - python manage.py shell
- ORM好处在哪里?
    - 1, 不需要编写sql,编写sql很繁琐
    - 2, 不需要关心使用的是什么数据库

11,添加数据(掌握)

- 注意点:
    - 1, 在django中所有和数据库的操作都是使用: 模型类.objects 开头的
    - 2, 增加数据两种方式

```
    # - 1.创建名为西游记的图书,添加到数据库中
    # book = BookInfo(btitle="西游记",bpub_date="1990-1-1")
    # book.save()

    # BookInfo.objects.create(btitle="红楼梦",bpub_date="1999-1-1") #等于上面两句话
```

12,基本查询(掌握)

- 目的: 掌握10条基本查询语句
- 注意点
    - 格式: BookInfo.objects.filter(字段__选项=值)
    - 特殊: exclude 排除
    - pk == id

13,FQ对象查询(理解)

- 目的: 掌握8条基本查询语句
- 注意点
- F,Q: 字段与字段之间的关系的查询(大小, 或,与 ,非)
- aggregate: 配合Sum,Max,Min等等聚合函数使用
- order_by: 排序, 降序是 -字段